OULUN AMMATTIKORKEAKOULU

Esa Juntura

KETTERÄT PERIAATTEET VERKKOPALVELUN OHJELMISTOKEHITYKSESSÄ

KETTERÄT PERIAATTEET VERKKOPALVELUN OHJELMISTOKEHITYKSESSÄ

Esa Juntura
Opinnäytetyö
Syksy 2019
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely tutkinto-ohjelma, web-ohjelmistokehitys

Tekijä: Esa Juntura
Opinnäytetyön nimi: Ketterät periaatteet verkkopalvelun ohjelmistokehityksessä
Työn ohjaaja: Anu Niva
Työn valmistumislukukausi ja -vuosi: Syksy 2019 Sivumäärä: 22

Tässä opinnäytetyössä on kehitetty edelleen olemassa olevaa verkkopalvelua. Vincit Oulu Oy:n asiakasyrityksessä verkossa tilattavien palveluiden tarjontaa lisättiin, joka asetti uusia vaatimuksia toiminnallisuuksille, järjestelmän hallinnalle ja käytettävyydelle. Asiakasyrityksen luvalla suunnitteluhankkeesta kirjoitettiin tämä opinnäyte. Verkkopalvelun uudet ominaisuudet toteutettiin osaksi olemassa olevaa järjestelmää, joka rajoitti joiltakin osin käytettävissä olevia ratkaisuja. Palveluihin liittyi erilaisia erillisiä järjestelmiä, joista osa oli kehitetty yrityksen omilla resursseilla omaa käyttöä varten ja osa oli valmisohjelmistoja. Toteutus vaati erilaisten ohjelmistorajapintojen toteutusta näiden järjestelmien ominaisuuksien hyödyntämiseksi tilattavissa palveluissa.

Uusien ominaisuuksien kehittäminen edellytti tietoa olemassa olevien järjestelmien toiminnasta ja rakenteesta. Tämän tiedon saatavuuteen vaikutti olemassa oleva dokumentaatio sekä niiden kehityksestä vastanneiden henkilöiden tietämys ja saatavuus. Kehitystyössä oli ketterän (Agile) ohjelmistokehityksen piirteitä. Ketterää kehittämistä hankaloittivat kehitettävän palvelun liittyminen suurempaan kokonaisuuteen, sekä ketterien periaatteiden vaillinainen toteutuminen työssä. Jatkossa ohjelmistokehityksen prosessissa olisikin hyödyllistä tutkia Lean-periaatteiden soveltamista, jotka on todettu käyttökelpoisiksi isommissa ohjelmistokehitysorganisaatioissa.

Työn tekijä perehtyi yrityksen ohjelmistokehityksen prosesseihin sekä käytettyihin teknologioihin, ja osallistui uusien ominaisuuksien suunnitteluun ja käytännön toteutukseen sekä käyttöliittymän että taustajärjestelmän osalta. Opinnäytetyön tuloksena verkkopalvelusta voitiin julkaista uusia ominaisuuksia sisältävä versio asiakaskäyttöön.

Asiasanat: ketterät menetelmät, ohjelmistokehitys, lean-ajattelu, verkkopalvelut

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Business Information Systems, Web application development

Author: Esa Juntura
Title of thesis: Agile methods in web applications development
Supervisor: Anu Niva
Term and year when the thesis was submitted: Fall 2019 Number of pages:22

This thesis concerns the further development of an existing web service. At a Vincit Oulu Oy customer organisation, there was a need to continue developing web-based subscription services. This posed new requirements for features, system management and usability. This thesis was based on this development project, with permission from customer. The new features were implemented on the existing (legacy) system, that somewhat limited the design choices. The subscription service had interfaces to several separate systems. Some of these had been developed internally and some were developed by third parties. The implementation of new features required new interfacing features to be developed in order to use these separate system parts.

The development of new features required knowledge of the existing systems' architecture and design. The availability of this information depended upon existing documentation and knowledge of the designers responsible for previous designs of the system. The development project work had features of agile development methods. The agile principles were only partially applied in the customer project development work. The application of agile methods was also somewhat difficult due to the connections to larger enterprise-level development plans.

In future, it could be fruitful to study the application of lean development principles in customer project processes. Lean principles have been found to address some of the shortfalls of agile methods in the development work done in larger organisations.

Thesis worker studied the software development processes and technologies used at the customer organisation. He took part in the planning and implementation of both back-end and front-end areas of the new features. As a result of the thesis work the customer organisation was able to publish a new version of the web service with new subscription features.

Keywords: agile methods, software development, lean thinking, web services

SISÄLLYS

1 JOHDANTO

Tietojärjestelmien kehittäminen voidaan jakaa uusien (”greenfield”-projektit) ja olemassa olevien järjestelmien kehittämiseen. Uusien järjestelmien kehittämisessä ei usein tarvitse ottaa huomioon muita tietojärjestelmiä, kuten olemassa olevien järjestelmien kehittämisessä.

Tässä opinnäytetyössä on kehitetty uusia ominaisuuksia olemassa olevaan verkkopalveluun. Vincit Oy:n asiakasyrityksessä verkossa tarjottavien tilauksellisten sisältöpalveluiden tarjontaa lisättiin, joka asetti uusia vaatimuksia palvelun toiminnallisuuksille, järjestelmän hallinnalle ja käytettävyydelle. Opinnäytetyön tekijä toimi ohjelmistokehittäjänä sisältöpalvelun uuden version kehityksessä. Tehtävänä oli suunnitella ja toteuttaa sekä web-käyttöliittymän että sisällön tuottavan taustajärjestelmän uudet ominaisuudet ja muutokset. Asiakasyrityksen luvalla työstä kirjoitettiin tämä opinnäytetyö.

Opinnäytetyön tavoitteena on kuvata työn eteneminen ja sen haasteet. Työn kuvauksessa ja arvioinnissa keskitytään erityisesti työn kulkuun ketterän kehittämisen (Agile Software Development, ASD) ja Lean-ajattelun periaatteiden kannalta. Lisäksi tarkastellaan, miten olemassa olevan järjestelmän piirteet vaikuttivat työhön ja sen etenemiseen.

2 KEHITTÄMISTEHTÄVÄN KUVAUS

Kun yritys tai muu organisaatio on ohittanut toiminnan aloitusvaiheen, toiminnassa yleensä on otettu käyttöön tietojärjestelmä tai -järjestelmiä. Usein ensin käyttöön otetut tietojärjestelmät käsittelevät organisaation kannalta olennaista tietoa, joten ne ovat toiminnalle arvokkaita (Avison & Fitzgerald 2002, 151-152). Organisaation toimintaa kehitettäessä myös tietojärjestelmien vaatimukset muuttuvat. Tällöin täytyy päättää, kehitetäänkö käytössä olevia järjestelmiä vai korvataanko ne uusilla. Käytössä olevien järjestelmien edelleen kehittämiseen liittyy useita haasteita, esimerkiksi vanhojen ohjelmointikielten osaajien puute ja alkuperäisen järjestelmän ratkaisuiden perusteiden epäselvyys (Thomas 2006, 19–20). Toisaalta vanhojen toiminnallisuuksien toteuttaminen uusilla teknologioilla voi olla taloudellisesti mahdotonta.

Tietojärjestelmien kehityksessä korostuu varsinkin telekommunikaatio- ja verkkosovellusten ja -palveluiden alalla tarve saada uudet tuotteet ja tuoteominaisuudet markkinoille nopeammassa tahdissa kuin ennen (Rodríguez 2013, 21). Markkinoiden paineen takia ohjelmistoteollisuus on etsinyt keinoja nopeuttaa ohjelmistokehitystä. Yksi eniten esillä olleista ratkaisuista on ollut ketterä ohjelmistokehitysmalli (Agile Software Development, ASD). ASD-menetelmät nojaavat neljään arvoon ja kahteentoista periaatteeseen, jotka on kuvattu Agile-manifestissa (Agile Manifesto, Beck ym. 2001, viitattu 9.8.2019).

Ketterän manifestin (Agile Manifesto, Beck ym. 2001, viitattu 9.8.2019) neljä arvoparia ovat

- Yksilöt ja vuorovaikutus ennen prosesseja ja työkaluja
- Toimiva ohjelmisto ennen perusteellista dokumentaatiota
- Yhteistyö asiakkaan kanssa ennen sopimusneuvottelua
- Muutoksiin vastaaminen ennen suunnitelman seuraamista

Manifestissa myönnetään, että oikean puoleisillakin asioilla on arvoa, mutta vasemmanpuoleisia arvostetaan manifestissa enemmän. Manifestin tekijät olivat kokoontuneet vuoden 2001 alussa miettimään vaihtoehtoja raskaaksi koetuille ohjelmistokehityksen menetelmille, jotka painottivat dokumentoinnin tärkeyttä.

Manifestin (Agile Manifesto, Beck ym. 2001, viitattu 9.8.2019) tekijät julkaisivat sivustollaan myös manifestin taustalla olevat kaksitoista periaatetta:

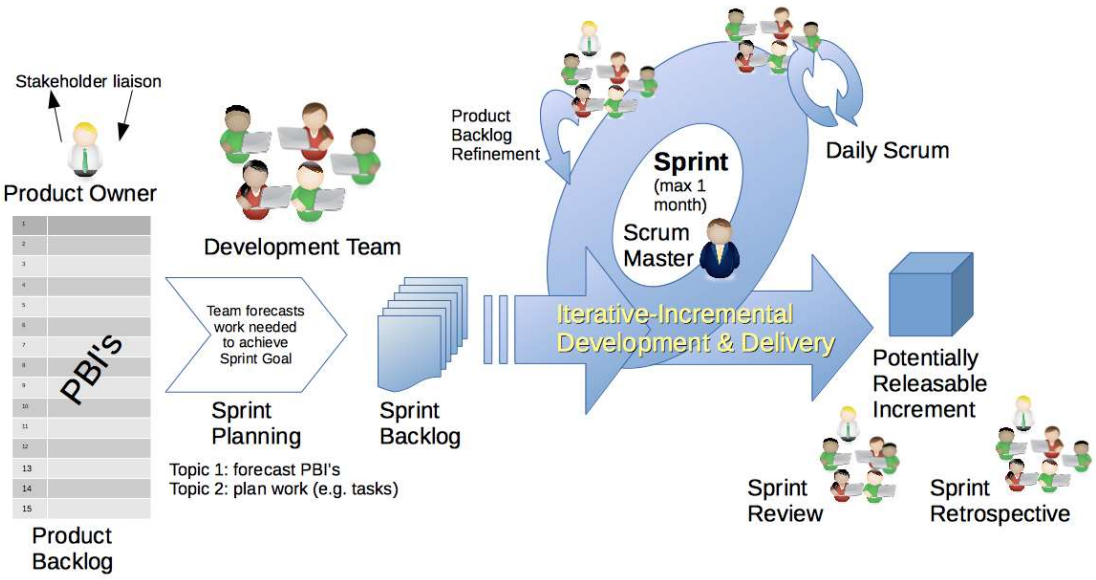
- Tärkein asia on asiakkaan tarpeiden täyttäminen nopeilla ja jatkuvilla ohjelmiston julkaisuilla (delivery), jotka tuovat arvoa.
- Muuttuvia vaatimuksia arvostetaan, myös myöhempänä kehityksessä. Muutokset valjastetaan tuomaan kilpailuetua asiakkaalle.
- Toimivia ohjelmistojulkaisuja tehdään usein, julkaisuväli kahdesta viikosta pariin kuukauteen, painottuen nopeampiin julkaisuihin.
- Liiketoiminnasta vastaavien täytyy työskennellä päivittäin yhdessä kehittäjien kanssa koko projektin ajan.
- Projektit perustetaan motivoituneiden yksilöiden ympärille, joille annetaan heidän tarvitsemansa ympäristö ja tuki. Yksilöiden kykyyn suoriutua työstä luotetaan.
- Tehokkain viestinnän muoto on keskustelu kasvokkain, jota käydään tiedon välitykseen projektille sekä projektin sisällä.
- Toimiva ohjelmisto on edistyksen tärkein mittari.
- Ketterät prosessit edistävät kestävää kehittämistä. Projektin tukijat, kehittäjät ja käyttäjät ylläpitävät tasaisen kehitystahdin.
- Jatkuva huomio teknisen tason erinomaisuuteen ja hyvään suunnitteluun parantaa ketteryyttä.
- Yksinkertaisuus, taito maksimoida tekemätön työ, on ensisijaista.
- Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvilta tiimeiltä.
- Tiimi tarkastelee säännöllisin väliajoin kuinka voitaisiin tulla tehokkaammiksi, ja säätää toimintaansa sen perusteella.

Ketterän kehityksen perusteita valotetaan Agile Alliancen sivustolla (Agile Alliance 2013, Viitattu 2.9.2019). Ketterässä kehityksessä painotetaan muita ohjelmistokehityksen lähestysmistapoja enemmän kehittämiseen osallistuvia ihmisiä ja heidän välistään viestintää. Ketterässä kehityksessä voidaan käyttää erilaisia ohjelmistokehityksen tapoja ja prosesseja, soveltaen niitä kehitystiimin tarpeisiin sopiviksi.

Yleisimpiin menetelmiin ketterässä kehityksessä kuuluu Scrum. Scrum-menetelmä on saanut alkunsa jo 1980-luvulla, ja sen tarkoituksena on pilkkoa isot kehitystehtävät pienempiin, helpommin hallittaviin kokonaisuuksiin. Lisäksi se painottaa kehitystiimin itsenäisyyttä sekä viestinnän tärkeyttä. Kuvassa 1 on kuvattu Scrum-prosessin eri osia ja päärooleja. Scrumissa noin kymmenen hengen tiimissä kehitystehtävä jaetaan yleensä noin 2 viikon jaksoissa, niinsanotuissa sprinteissä,

valmistuviin osiin. Ennen sprintin aloitusta sen tavoitteet suunnitellaan yhteisessä palaverissa, ja sen valmistuttua tulokset käydään läpi katselmointi- ja retrospektiivi-palavereissa. Retrospektiivin tarkoituksena on myös oppia virheistä ja selvittää, miten seuraavassa sprintissä voitaisiin onnistua paremmin. Scrumissa tehtävien etenemistä seurataan päivittäisissä lyhyissä, noin 15 minuutin kokouksissa. Näissä jokainen tiimin jäsen kertoo, mitä on saavuttanut edellisenä päivänä tiimin tavoitteeseen pääsemiseksi, mitä aikoo saada tänään valmiiksi, ja onko havainnut ongelmia, jotka estävät tiimin tavoitteeseen pääsyä. (Schwaber & Sutherland 2016, 11.)

Scrumissa sprintin tehtävät poimitaan suunnittelupalaverissa listalta, jota kutsutaan englanniksi nimellä backlog. Listaa ylläpitää tuotteen omistaja (engl. Product Owner), joka vastaa myös viestinnästä varsinkin asiakkaisiin päin. Hänellä on tuotteen liiketoimintavastuu. Tiimin työskentelyn tehokkuudesta vastaa Scrum master, joka pyrkii suojamaan tiimin työskentelyä häiriöiltä. Hän myös vastaa, että työskentely sujuu Scrumin periaatteiden mukaan. (Schwaber ja Sutherland 2016, 6.)



KUVA 1: Scrum-prosessi, roolit ja toiminnat. Lähde: Dr Ian Mitchell - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44894952. Muuttamaton versio.

ASD-menetelmistä on tutkitusti ollut hyötyä pienissä kehitystiimeissä, mutta suurempien kokonaisuuksien kehittämisessä suuremmissa organisaatioissa niiden käytössä on ollut haasteita (Poppendieck ja Poppendieck, 2003). Näiden parempaa hallintaa varten on ohjelmistokehityksen prosesseihin otettu viime aikoina mukaan myös Lean-ajattelun mukaisia toimintamalleja. Lean-ajattelun juuret ovat japanilaisessa autoteollisuudessa, ja sen pääperiaatteina ovat työn tuottaman

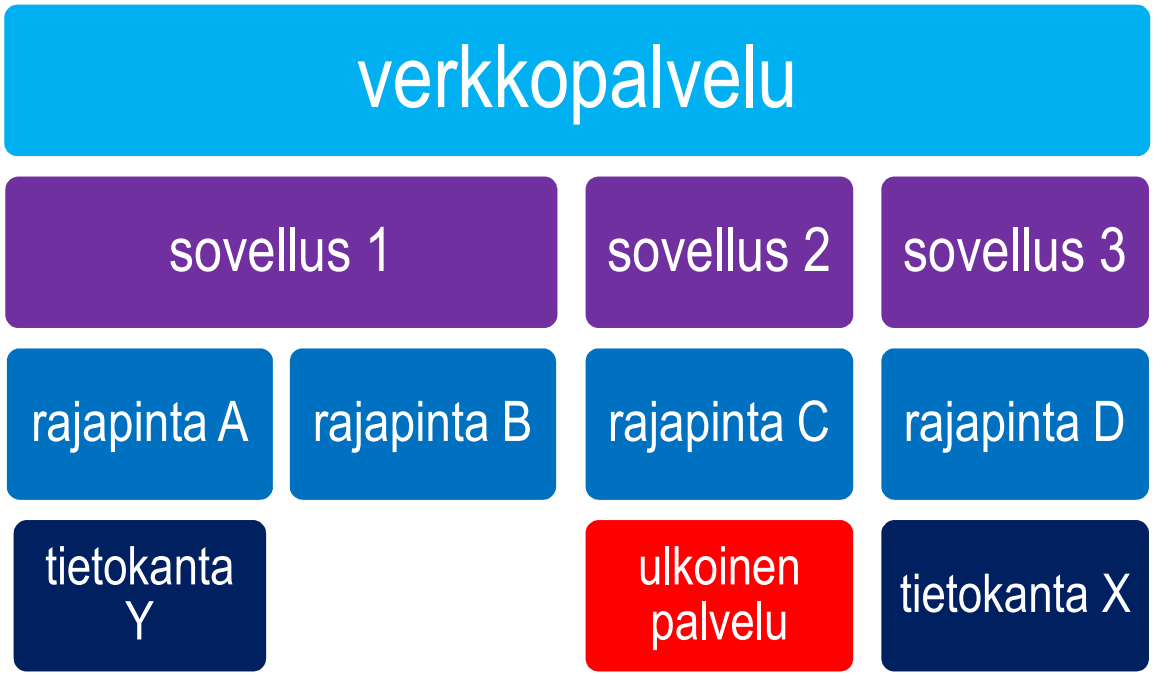
arvon maksimointi ja hukan minimointi. Tärkeinä konsepteina ovat arvo asiakkaalle, arvoketju, hukan vähentäminen, työn kulun analysointi ja jatkuva parantaminen. Lean-ajattelun ensimmäisiä esille tuojia olivat Womack ja Jones (Womack ja Jones, 1996).

ASD-menetelmiä sekä Lean-ajattelua onkin pyritty yhdistämään ohjelmistokehityksen tehokkuuden parantamiseksi. Lean-ajattelu täydentää ASD-menetelmiä erityisesti kokonaisvaltaisella lähestymistavalla (end-to-end) sekä flow-ajattelulla. Näissä on kuitenkin vastakkaisia pyrkimyksiä, Agilen pyrkimys joustavuuteen voi aiheuttaa hukkatyötä, kun ohjelmistokehityksen tavoitteita muutetaan kehitysprojektin aikana asiakastarpeiden muututtua tai tarkennuttua. Käytännössä näitä periaatteita kuitenkin voidaan painottaa ja soveltaa kussakin organisaatiossa ja projektissa omien tarpeiden mukaisesti. Eräässä esimerkkitapauksessa painotettiin (asiakas)arvoa ja laatua hukkaa enemmän. Ohjelmistokehityksessä hukkana voidaan pitää viivästyneitä päätöksiä, keskeneräistä työtä sekä käyttämättömiä ominaisuuksia (Rodríguez, 2013.)

Tässä työssä käsitellyn sisältöpalvelun ensimmäinen versio oli julkaistu uutena osana asiakasyrityksen verkkopalvelussa vuoden 2018 lopussa, ja uuden version toiminnallisuuksia oli jo tiedossa tässä vaiheessa. Ensimmäisessä versiossa asiakkailla oli mahdollisuus kokeilla verkossa tarjottavia sisältöjä ennen maksullisen tilauksen tekemistä. Kokeilusisältö ja maksullisen tilauksen sisältö olivat samat. Uuden sisällön tarjoaminen edellytti muutoksia sekä taustajärjestelmään (engl. back end) että käyttöliittymään (engl. front end). Osa muutoksista tehtiin, jotta uusia tuotteita voitaisiin jatkossa lisätä helpommin, eli tuotteiden hallinnan helpottamiseksi. Tietoturvaa haluttiin myös parantaa. Palvelun ulkoasua ja toiminnallisuutta uudistettiin sopimaan paremmin yhteen muiden yrityksen tarjoamien verkkopalveluiden kanssa. Projektiin osallistui pääasiassa kolmen eri tiimin jäseniä, jotka kaikki työskentelivät samassa rakennuksessa. Tämä helpotti yhteistyötä, ja oli osaltaan mahdollistamassa ketterää kehitystä.

3 KEHITTÄMISTEHTÄVÄN ETENEMINEN

Projekti aloitettiin tammikuun 2019 alussa. Ensimmäisellä viikolla pidettiin perehdytys yrityksen verkkopalveluiden osista, rakenteesta ja toiminnasta. Kuvassa 2 on esitetty palvelun rakenteen periaatekuva. Toisella työviikolla varmistui, että työ tulisi koskemaan työskentelemään tilattavien sisältöjen kehitystä. Samalla tutustuttiin tarkemmin tilattavien sisältöjen palveluun. Uusista toiminnoista oli jo käyttöliittymäluonnoksia. Varsinaisen sisältöosan käyttöliittymän ulkoasun ja toiminnallisuuden luonnosteli palvelumuotoilija. Käyttöliittymän toteutus tehtiin tämän suunnitelman pohjalta. Näiden suunnitelmien perusteella tehtiin ensimmäisistä tehtävistä työsuunnitelma työmääräarvioineen.



KUVA 2: Periaatekuva verkkopalvelun rakenteesta

3.1 Teknologiat

Teknologiavalinnat tehtiin yrityksessä jo käytössä olevista teknologioista. Valintoihin vaikutti paljon se, mihin osaan olemassa olevaa palvelua uudet ominaisuudet vaikuttivat. Esimerkiksi käyttöoikeuksien hallinta tapahtui verkkopalvelun käyttäjähallinta-sovelluksessa, joka oli toteutettu ASP.NET Web Forms-alustalle. Ohjelmointikielenä oli tässä VisualBasic.NET. Myös toteutukseen käytettävissä oleva aika vaikutti teknologiavalintoihin. Uusien teknologioiden arviointi ja kokeilu

ennen varsinaista toteutusta olisi vienyt enemmän aikaa, kuin vanhalle alustalle tekeminen. Lisäksi kaiken vanhan olemassa olevan toiminnallisuuden toteuttaminen uudelle alustalle olisi vaatinut paljon työtä.

Palvelun sisältöosuuden käyttöliittymäsovellus toteutettiin aiempaa AureliaJS-toteutusta täydentäen. Käyttöliittymän ohjelmointikielenä oli TypeScript. Taustajärjestelmä käytti Microsoft .NET Core 2.x -alustaa, ohjelmointikielenä C#. Käyttöliittymäsovelluksen palvelinsovelluksessa ei ollut tietokantayhteyksiä, vaan se käytti tiedonhakuun kahta eri REST API -rajapintaa. Toinen rajapinta oli yleisrajapinta käyttäjätunnistuksiin ja muihin verkkopalvelun eri sovellusten tarvitsemiin toimintoihin. Toinen rajapinta oli sisältöpalvelun tiedonhakua ja käsittelyä varten. Myös REST API -rajapinnat oli toteutettu .NET Core 2.x alustalle C#-kielellä. Tietokantoja käytettiin uudemmissa sovelluksissa Entity Framework (EF) Core -kirjastojen tuella, vanhemmissa oli suoria SQL-kyselyitä. Tietovarastoista osa oli yrityksen omassa laitteistossa olevia SQL Server -tietokantoja, osa Microsoftin Azure-palvelussa olevia tietokantoja. Azure-palvelussa olevaa dataa käytettiin REST-rajapintojen kautta. Käyttäjien tunnistus tehtiin keskitetysti identiteetin hallintapalvelimen avulla.

Lähdekoodin versionhallinta oli toteutettu Atlassian Bitbucket/Stash -palvelulla, joka toimi yrityksen omalla palvelimella. Palvelu oli rakennettu Git-versionhallinnalle. Palvelua saattoi käyttää asiakasohjelman kautta tai suoraan komentorivin Git-komennoilla. Lähdekoodin hallinnassa oli käytössä eri versiohaarat (branch). Pääkäytäntönä oli, että uudet ominaisuudet ja korjaukset tehdään omiin haaroihin, ja integroidaan (merge) valmiina päähaaraan. Päähaarasta viedään säännöllisesti versiot testaukseen ja sen jälkeen tuotantoon.

3.2 Prosessityökalut ja työtavat

Ohjelmistokehityksen tehtäviä hallinnoitiin Atlassian Jira-alustalla toimivassa palvelussa. Uusien työsuunnitelmien ja muun uuden dokumentaation pääasiallinen sijainti oli Atlassian Confluence (wiki)-palvelu, mutta dokumentaatiota oli myös vanhemmassa Microsoft Sharepoint -pohjaisessa järjestelmässä sekä suoraan verkkolevyillä. Järjestelmien tietojen välillä ei ollut juuri linkitystä, ja osa tiedoista oli vanhentunutta. Järjestelmään liittyvien tietojen haussa ei voinut näistäkään syistä suunnittelun aikana nojata täysin sähköiseen dokumentaatioon, vaan iso osa työn vaatimasta tiedosta oli hankittava henkilökohtaisesti muilta kehittäjiltä kysymällä ja keskustelemalla. Dokumentaation puutteellisuus ja vanhentuminen ovat yleisiä ilmiöitä ohjelmistokehityksessä

(esimerkiksi Thomas, 2006, s. 5). Tässä palvelun eri sovellusten kehitystiimien sijainti vierekkäin samassa kerroksessa auttoi suuresti. Eri tiimiin siirtynyt tilattavien sisältöpalveluiden kehittämisestä aiemmin vastannut kehittäjä antoi arvokasta tietoa sovelluksen jatkokehittämistä varten.

Kehitystyö oli jaksotettu kahden viikon jaksoihin eli sprintteihin. Kehitystyötä oli organisoitu Scrum-menetelmän (esimerkiksi Rodríguez 2013, 46) tyyppisesti. Tyypillisestä Scrum-työskentelystä työskentely poikkesi siten, ettei varsinaista Scrum-masteria ollut nimetty. Kunkin sprintin alussa käytiin läpi tulevan kahden viikon kehitystavoitteet. Nämä liittyivät useimmiten Confluence-palveluun tallennettuihin tuotemäärittelyihin ja käyttöliittymäluonnoksiin. Tavoitteet tallennettiin kehitystehtävinä Jira-palveluun. Tehtävien valmistumisen jälkeen ne kuitattiin palvelussa valmiiksi. Erillistä sprintin lopetuspalaveria ei aina pidetty, vaan sprintin tulokset käytiin vaihtelevalla tarkkuudella läpi uuden sprintin aloituspalaverissa. Kehitystyössä kommunikointi tapahtui päivittäisissä aamupalavereissa, joissa kukin osallistuja kertoi edellisen päivän tekemisen tuloksista ja kuluvan päivän tavoitteista ja ongelmista. Tämä vastasi päivittäistä Scrum-palaveria (engl. daily Scrum).

Kahden viikon välein testattuja ominaisuuksia ja korjauksia julkaistiin tuotantoon. Sovellusten integraatio- ja asennus oli automatisoitu jatkuvan integroinnin periaatteen mukaisesti (continuous integration, CI). Kaikille sovelluksille ei ollut toteutettu automaattista (yksikkö tms.) testausta, vaan esimerkiksi sisältöpalvelun testaus tehtiin käyttöliittymän kautta testausympäristössä. CI-järjestelmän olemassaolo ja ylläpito mahdollistivat osaltaan ketterän kehityksen periaatteiden mukaisen nopean julkaisun (Beck ym. 2001, Viitattu 9.8.2019.)

3.3 Käyttöoikeuksien hallinnan toteutus

Ensimmäisenä kehitettävänä kohteena oli käyttäjien käyttöoikeuksien hallinta. Uusien tilattavien sisältöjen tullessa näkyville palveluun haluttiin sisältöjen näkyvyyttä hallinnoida käyttäjäkohtaisesti. Toiminnosta oli jo tehty tietomallin päivityssuunnitelma, ja tämän perusteella pystyttiin aloittamaan kehitys.

Tietomalliin lisättiin linkitys käyttäjän tunnuksen sekä tilattavan sisällön välille. Tämä edellytti uuden tietokantataulun luontia tietokantaan. Ensin tämä tehtiin käsin, mutta projektin edetessä selvisi, että taulujen luonti ja päivitys EF Core-ympäristössä oli hyödyllisintä tehdä ns. tietokantamigraatioiden avulla. Tällöin EF Core-skriptin avulla voidaan päivittää tietokannan rakenne vastaamaan

sovelluskoodin tietorakennetta (oliot), jolloin tietomalli ja kannan rakenne varmasti vastaavat toisiaan. Migraatioiden käyttö oli organisaatiossa uudehko käytäntö, eikä siitä ollut selkeää ohjeistusta. Migraatioille tosin ei edes ollut valmista tukea vanhemmalle .NET Framework - alustalle tehdyissä sovelluksissa, jotka eivät käyttäneet esimerkiksi Entity Framework-kirjastoja. Migraatioiden käyttöön siirryttiin huhtikuussa.

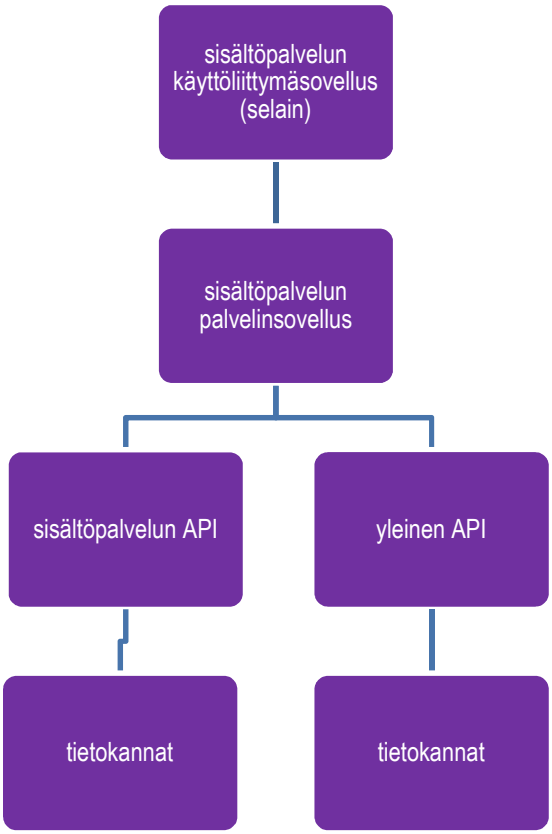
Käyttöoikeustietojen tallennuksen jälkeen alettiin toteuttamaan käyttöoikeuksien tarkistusta sisällön näyttämisessä. Tämä koodattiin sisältöpalvelun taustajärjestelmään. Taustajärjestelmä ja käyttöliittymä viestivät REST API-rajapinnan kautta. Oikeuksien tarkistus tehtiin siinä vaiheessa, kun käyttäjälle haettiin listaa näytettävistä sisällöistä. Jos käyttäjällä ei ollut johonkin sisältöön oikeuksia, API ei palauttanut käyttöliittymään linkkiä tähän sisältöön. Myöhemmässä vaiheessa lisättiin myös oikeuksien tarkistus sisällön tietokantahakuun. Tällä pyrittiin estämään pelkän sisältölinkin avulla tietoihin pääseminen.

Tässä vaiheessa (tammikuun loppu) pystyttiin lisäämään tietokantakäskyillä käyttäjille oikeuksia sisältöihin, ja käyttöliittymässä näytettiin vain ne sisällöt, joihin käyttäjillä oli oikeudet. Jotta palvelua voitaisiin oikeasti käyttää, oli myös käyttäjähallintaan lisättävä mahdollisuus hallinnoida sisältöoikeuksia. Tällöin asiakkaat pystyisivät itse hallinnoimaan eri käyttäjien oikeuksia. Tämän ominaisuuden myötä palvelun uuden version voisi julkaista asiakkaiden käyttöön.

Käyttäjien tietojen hallinnointi oli toteutettu ASP.NET Web Forms-alustalla tehdyssä web-sovelluksessa, jonka ohjelmointikieli oli VisualBasic.NET. Ensin toteutettiin olemassa olevan sisällön oikeuksien hallintaan käyttöliittymän komponentit. Käyttöliittymään lisättiin oma näkymä sisältöoikeuksien hallinnalle, jossa oli sisältökohtaiset valintapainikkeet (radio button) sekä koko palvelun näkyvyyden valintapainike. Sisältökohtaiset painikkeet eivät olleet näkyvillä, mikäli käyttäjälle ei ollut valittu palvelua näkyville palvelun valintapainikkeesta. Jos käyttäjälle oli lisätty oikeuksia, mutta palvelu poistettiin näkyvistä, käyttäjälle annetut oikeudet poistettiin tietokannasta. Hallintanäkymä tuli näkyviin asiakasyrityksen pääkäyttäjälle, mikäli asiakasyritys oli tilannut palvelusta sisältöä.

Hallintanäkymää suunniteltaessa ja toteutettaessa kävi koodia tutkimalla selville, että koko palvelun käyttäjäkohtainen näkyvyystieto, sekä palvelun tilaustiedot talletettiin eri tietokantoihin kuin käyttäjien palvelukohtaiset käyttöoikeudet. Edelliset tiedot haettiin suoraan tietokantahauilla, jälkimmäisten hakuun käytettiin sisältöpalvelun API-rajapinnan sovellusta. Tätä varten

hallintasovellukseen toteutettiin sisältöpalvelun API-rajapinnan yhteys. Suunnitteluvaiheessa yhtenä vaihtoehtona oli toteuttaa yhteydet vain yleiseen API-rajapintaan, joka sitten olisi taas voinut kutsua tarvittaessa sisältöpalvelun API-rajapintaa. Tässä vaiheessa sisältöpalvelun rajapintaa ei kutsuttu käyttäjähallinnan sovelluksesta. Lopuksi päädyttiin kuitenkin käyttämään kahta erillistä rajapintaa, jotta saman aihepiirin kutsuja ei olisi useassa paikassa. Valintaa tehtäessä käytiin keskusteluja kehittäjien kesken, tällaisista suunnitteluratkaisuista ei ollut kirjoitettua ohjeistusta. Kuvassa 3 on esitetty sisältöpalveluun liittyvät sovellukset ja rajapinnat.



KUVA 3: Sisältöpalvelun sovellukset ja rajapinnat

Sisältöjen katseluoikeuksien tallentamisessa käytettiin sisältöpalvelun API-rajapinnan EF Core-kirjastoja. Kävi ilmi, että tietokantayhteyksien toiminnallisuus oli toteutettu projektin ensimmäisessä vaiheessa vain osittain, eikä esimerkiksi tietojen tallennus toiminut. Tästä koitui jonkin verran yllättävää lisätyötä. Lopullinen toteutus valmistui vasta loppukevään sovellusversioon. Aiemman toteutuksen keskeneräisyys viittasi myös ketterissä projekteissa usein kertyvään ns. tekniseen velkaan (Allman, 2012). Teknisen velan käsite tarkoittaa työtä, joka vaaditaan myöhemmin ohjelmistoa kehitettäessä, koska aiemmin on valittu toteutettavaksi helpommin ja/tai nopeammin

toteutettavia ratkaisuja. Myös esimerkiksi jäljempänä kuvattava uusien sisältöjen muuttaminen dynaamiseksi voidaan ymmärtää teknisen velan maksamisena.

Käyttäjäoikeuksien hallinnan kokonaisuus oli käytännössä valmis ja testattu maaliskuun loppuun mennessä. Sitä ei kuitenkaan julkaistu tuotantoon, koska sisältöpalvelun julkaisuun haluttiin muitakin ominaisuuksia.

3.4 Sisältöjen tilaaminen

Helmikuun alkupuolella tuli myös sisältöjä valmistelevalta tiimiltä lisätietoa uusista julkaistavista sisällöistä. Näihin liittyen asiakkaille tulisi myös eri sisältövaihtoehtoja tilattavaksi, eli verkossa tarjottavien sisältötuotteiden määrä kasvaisi. Ensimmäisessä palvelun versiossa tuotteiden tiedot olivat suoraan ohjelmakoodissa (niinsanotusti ”kovakoodattuja”), joten tässä vaiheessa olisi hyvä viimeistään miettiä dynaamisempaa ratkaisua. Tällöin tuotteita voitaisiin hallita ilman koodimuutoksia (ja uuden version testausta ja julkaisua).

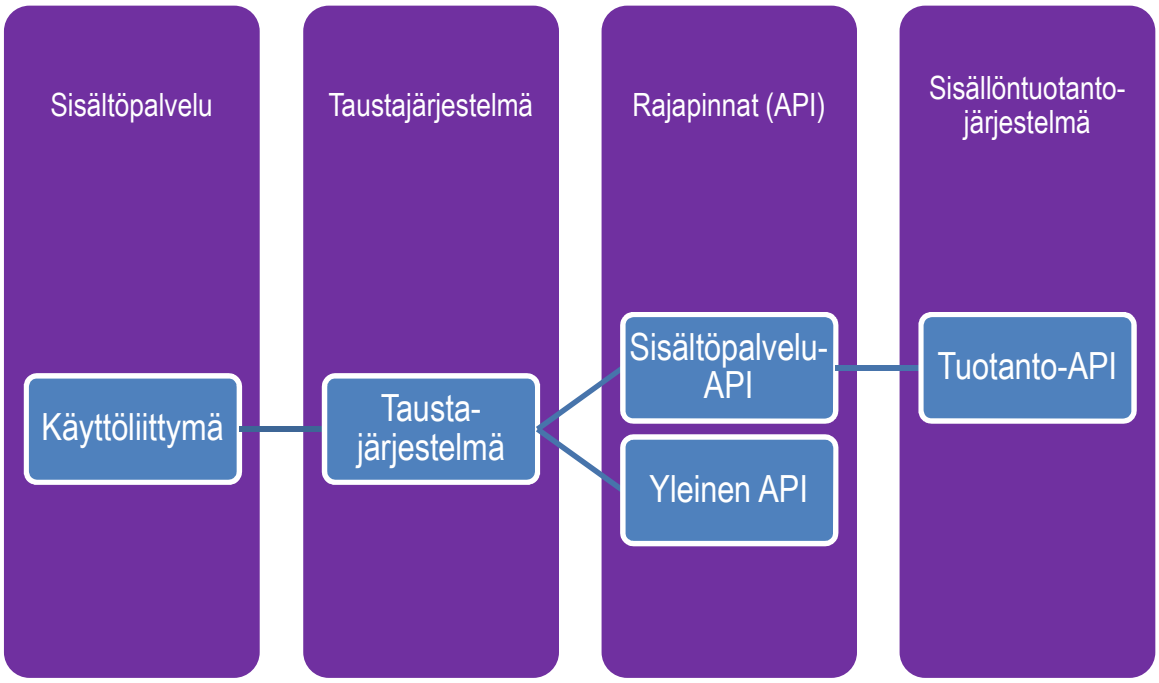
Sisältötuotteiden rakennetta alettiin suunnitella. Eri tuotteissa tulisi olemaan erilaisia sisältöjä ja hintoja. Eri tuotteet tulisivat eri asiakasryhmille. Pitkään kehityksen aikana oli epäselvää, tulisiko hinnoittelusta sisältökohtaista vai tuotekohtaista. Tämän takia tietomalliin tehtiin useaan kertaan muutoksia kevään aikana. Lopulta päädyttiin tuotekohtaiseen hinnoitteluun. Ensimmäinen palaveri eri sidosryhmien kesken, jossa oli myös kehittäjiä mukana, käytiin huhtikuun alussa. Tässä koostettiin tietoja sisältöpalvelun asiakasvaatimuksista sekä yhteyksistä muihin palveluihin. Tässä tuli myös tarkempaa tietoa aikatauluvaatimuksista. Asiakasyrityksen merkittävän palvelun julkaisun takia sisältöpalvelun päivityksen tavoiteaikataulu oli alkukesä 2019.

Uusien sisältöjen näyttäminen vaati myös päivitystä sisältöpalvelun käyttöliittymän navigaatiotoimintoihin. Käyttökokemus (user experience, UX) -suunnitelmien perusteella käyttöliittymään lisättiin muun palvelun kanssa yhtenäinen sivupalkki (AureliaJS- alustalla, Typescript- ohjelmointikieli), jossa oli dynaamiset linkit eri sisältötuotteisiin. Linkit luotiin tarjolla olevien tuotteiden sekä palvelua käyttävän yrityksen ja käyttäjän tietojen perusteella.

Maaliskuun lopussa tilaustoiminnallisuutta toteuttamaan otettiin toinen kokenut kehittäjä. Hänen aiempi kokemuksensa asiakasyrityksen tuotteiden tietorakenteista ja niiden välisistä linkeistä nopeutti kehitystä. Tämä oli tarpeen myös julkaisuaikataulussa pysymiseksi.

3.5 Uudet sisällöt

Maaliskuun loppupuolella alkoi työ uusien sisältöjen parissa. Käyttöliittymään tarvittiin uusia toiminnallisuuksia näiden näyttämiseksi. Sisällöt luotiin kolmannen osapuolen kehittämässä järjestelmässä, josta ne saatiin API-rajapintojen kutsujen avulla sisältöpalveluun. Sisältöpalvelun ja sisällöntuotantojärjestelmän välinen yhteys on esitetty kuvassa 4. Sisällöntuotantojärjestelmän opiskelu vaati jonkin verran aikaa. Samalla suunniteltiin ja toteutettiin parannukset sisällön haun tietoturvaan.



KUVA 4: Sisältöpalvelun linkitys sisällöntuotantojärjestelmään

Sisältöjä haettaessa taustajärjestelmään välitettiin joitakin parametrejä, joiden perusteella haettiin oikeat halutut tiedot sisältöön. Uudet sisällöt vaativat uusien parametrien lisäämistä. Näille tarvittiin käyttöliittymään komponentit. Parametrien mahdolliset arvot riippuivat käyttäjän tiedoista, joten ne oli haettava käyttäjäkohtaisesti taustajärjestelmästä. Tämä vaati asiakastiedon API- rajapinnan käyttöä. Asiakastietoa voitiin hakea kahdesta eri yrityksen palvelusta erillisten API- rajapintojen kautta, eikä ensin ollut selvää, kumpaa voisi käyttää. Toisen käyttö vaati suurehkon aiemmin tehdyn koodikirjaston käyttöönottoa, toinen taas uuden API-yhteyden toteutusta. Ensimmäinen vaihtoehto oli nopeampi toteuttaa, joten kehittäjä valitsi ja toteutti tiedon haun ensin tällä ratkaisulla. Tämän jälkeen kuitenkin tuli ilmi, että aiemmin tehdyt koodikirjastot tuovat liikaa turhaa koodia ja riippuvuuksia sovellukseen, joten toteutus tehtiin uudestaan toiselle rajapinnalle. Tämä vaati

hieman enemmän suunnittelua, mutta lopulta kyseessä oli vain n. 1-2 päivän työ. Web-palvelun eri osien vanhemmillakaan suunnittelijoilla ei ollut tietoa palvelun eri osissa käytettäväksi sovituista ratkaisuista. Aina näitä ei ollut selkeästi sovittukaan, vaan ratkaisut tehtiin tapauskohtaisesti. Mikäli suunnittelusta olisi järjestetty kokous, asia olisi ratkennut ilman 1-2 päivän ylimääräistä työtä.

Uusien sisältöjen dynaamisuus asetti uusia vaatimuksia myös käyttöoikeuksien hallinnan käyttöliittymälle. Sisältökohtaiset valintapainikkeet piti myös toteuttaa dynaamisina, eli painikkeiden tiedot piti hakea taustajärjestelmästä, kun oikeuksien hallinnan käyttöliittymä avattiin. Vanhempaa teknologiaa edustava ASP.NET Web Forms-alusta asetti tämän toteutukselle omat rajoitteensa. Web Forms-alusta vaatii web-sivun latauksen uudelleen minkä tahansa sisällön päivittämiseksi, jos ei käytetä erillisiä Javascript-liitännäisiä. Uusien liitännäiskomponenttien käyttäminen taas olisi monimutkaistanut koodia. Toteutus vei 3 päivää, kun AureliaJS-kirjastoa hyväksi käyttävällä käyttöliittymällä tämä olisi ollut todennäköisesti 1-2 päivän työ.

Sisältötuotannon taustajärjestelmä päätettiin myös päivittää uudempaan versioon ennen palvelun julkaisua. Taustajärjestelmästä oli vain tuotantoversio asennettuna, tähän asti kehitys ja testaus oli käyttänyt samaa taustajärjestelmää kuin julkaistu tuotantoversiokin. Vanhan version päivitys uuteen ei onnistunut suoraan. Uusi versio oli asennettava uudelle palvelimelle. Edellisen asennuksen dokumentointi oli puutteellista, ja oikeiden asetusten selvittäminen ja kokeilu vei aikaa, noin 2-3 päivää. Tämä aiheutti odottamattoman viivästyksen testaukseen ja julkaisuun. Mikäli taustajärjestelmälle olisi ollut asennettuna jatkuvan integraation ja asennuksen toiminnot (CID), olisi ketterämpi kehitys ollut mahdollista myös tämän osalta.

4 Pohdinta

Tässä työssä kuvatun kehityshankkeen tavoitteena oli kehittää edelleen olemassa olevaa verkkopalvelua ja lisätä siihen uusia toiminnallisuuksia. Näitä olivat sisältöpalvelun käyttöoikeuksien hallinta, uudet sisällöt ja niiden tuoterakenteet sekä tilaustoiminnallisuus. Kesän 2019 alkuun mennessä suunnitelluista toiminnallisuuksista oli toteutettu alun perin suunnitellut ominaisuudet. Sovellus ei kuitenkaan ollut vielä kypsä julkaistavaksi loppuasiakkaille, koko järjestelmää ei ehditty testaamaan lopullisessa kokoonpanossa, ja joitakin toiminnallisuuksien puutteita oli vielä korjaamatta.

Olemassa oleva järjestelmä vaikutti työhön monin tavoin. Lähes kaikkiin suunnittelussa tehtyihin valintoihin vaikuttivat aiemmin tehdyt suunnitteluratkaisut. Osa työajasta meni aiempien toteutusten periaatteiden selvittämiseen. Dokumentaation puutetta paikkasi järjestelmää aiemmin suunnitelleiden henkilöiden panos. Vanhat teknologiat rajoittivat joiltakin osin uusien toimintojen sisältöä ja toimintaa, ja hidastivat kehittämistä. Sisältöjä tuottavan taustapalvelun vanhan version päivitys vei aikaa suunnittelulta ja testaukselta.

Työskentely tapahtui osana tiimiä, jonka työskentelyssä käytettiin joitakin ketterän kehityksen periaatteita (Beck ym. 2001, Viitattu 9.8.2019). Näitä olivat muuttuvien tuotevaatimusten hyväksyminen, kommunikointi keskustelemalla, turhan työn minimointi, itseorganisoituva tiimi ja pyrkimys tiheään julkaisusykliin. Kehityksessä kiinnitettiin myös huomiota laatuun ja tekniseen pätevyyteen. Agile-manifestin periaatteista vähemmälle jäi tehdyn työn arviointi, ja asiakasarvon tuottaminen valmiilla julkaisulla jäi vielä toteutumatta kesällä 2019. Osaksi julkaisun viivästyminen johtui taustajärjestelmän päivittämien tuomasta lisätyöstä. Toisaalta jälkikäteen ajatellen muutamien suunnitteluratkaisujen uusiminen vei 1-2 viikkoa työaikaa projektista. Lisäksi projektissa oli vain yksi kehittäjä huhtikuuhun asti. Lopullinen julkaisu asiakkaille tapahtui kesän 2019 jälkeen.

Projektitiimi ei toiminut koko aikaa tiiviissä yhteistyössä, vaan välistä kehittäjille jäi epäselväksi tuoteomistajan ja sisäisen asiakkaan ja muiden sidosryhmien tekemien päätösten perusteet. Jotkin kehitettävän järjestelmän osista eivät olleet yleisen CID-järjestelmän piirissä, mikä auttaisi nopeiden julkaisujen teossa. Ketterän kehityksen mallien soveltamisessa olisi siis monilta osiltaan

vielä kehitettävää asiakasyrityksessä. Yhteenveto ketterän kehityksen manifestin periaatteiden toteutumisesta on taulukossa 1.

TAULUKKO 1: Ketterän kehityksen periaatteet (Beck ym 2001, Viitattu 9.8.2019) ja niiden toteutuminen kehityshankkeessa

PERIAATE	TOTEUTUMINEN	KOMMENTIT
Asiakasarvon tuottaminen aikaisilla ja jatkuvilla ohjelmiston julkaisuilla	-	julkaisua ei tullut 6 kk aikana
Muutosten mukaan ottaminen myös myöhäisessä vaiheessa	hyvin	
Ohjelmiston julkaisu usein, mielellään viikkojen välein	-	julkaisua ei tullut 6 kk aikana
Liiketoiminnan ja kehittäjien päivittäinen yhteistyö	osittain	liiketoiminta mukana osan aikaa
Motivoituneiden yksilöiden tukeminen ja luottamus	hyvin	ei ulkoa päin tulevaa sanelua suunnittelussa
Tehokas henkilökohtainen viestintä kasvokkain	osittain	suunnittelussa ei aina koko tiimi mukana
Toimiva ohjelmisto on etenemisen mitta	osittain	Testaus tapahtui viiveellä, julkaisun viivästyminen
Tasaisen kehitysvauhdin ylläpitäminen	osittain	julkaisua ei tullut 6kk aikana
Erinomaisen teknisen tason ja suunnittelun ylläpitäminen	osittain	muutoksia ei aina katselmoitu tai tasoa arvioitu
Työn määrän minimointi yksinkertaistamalla	hyvin	
Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvilta tiimeiltä	osittain	ei juuri ulkoa päin tulleita vaatimuksia; muut projektit kuitenkin vaikuttivat
Tekemisen tehokkuutta arvioidaan ja parannetaan säännöllisesti	osittain	arviointi satunnaista

Sisältöpalvelun kehitysprojektin riippuvuudet yrityksen muihin tuotteisiin ja niiden kehittämiseen toivat esille, minkälaisia haasteita yhtä tiimiä suuremmassa organisaatiossa syntyy Agile-periaatteiden noudattamiselle (esimerkiksi Maples, 2009). Yhtä tuotetta ei voi enää suunnitella

ilman toisten tuotteiden kehityksen huomioon ottamista. Tämä vaatii kehitystiimien ja liiketoimintavastuullisten yhteistyötä. Tällaisessa tilanteessa Lean-periaatteiden soveltamisesta ohjelmistokehitykseen voisi olla hyötyä. Agile- ja Lean-periaatteiden soveltamisesta laajemmissa organisaatioissa on myös kaupallistettuja malleja, mm. SAFe (Overview of the Scaled Agile Framework for Lean Enterprises, 2019).

Opinnäytetyön aikana kehitin omaa osaamistani laaja-alaisesti. Erilaisista Microsoftin verkkosovellusteknologioista opin lisää ASP.NET Web Forms:sta, .NET Core:sta sekä ASP.NET MVC:stä. Web-käyttöliittymäohjelmoinnissa eniten uutta opin perehtyessäni AureliaJS:n.

Tietokantaliittymien osalta pääsin tutustumaan .NET Core Entity Framework-teknologioihin, jossa tuli ensimmäistä kertaa käytettyä ORM (object relational mapping) - menetelmiä. Työn aikana syvensin myös ketterän kehityksen taustan ja teorian tuntemusta.

Työn tekemisen aikana kehitettiin asiakasyrityksen verkkopalveluun uusia ominaisuuksia. Näitä olivat

- Sisältöjen käyttöoikeuksien käyttäjäkohtainen hallinta
- Uudet tilattavat sisällöt
- Sisältöjen tilaustoiminnallisuudet
- Sisältöpalvelun web-sovelluksen käyttöliittymän navigaation päivitykset

Uusia toiminnallisuuksia varten tein muutoksia asetusten ja sisältöpalvelun web-sovellusten lisäksi myös kahteen eri taustarajapintaan (REST API).

Opinäytetyön tuloksena verkkopalvelusta voitiin julkaista uusia ominaisuuksia sisältävä versio asiakaskäyttöön. Opinnäytetyön päättymisen jälkeen jatkoin asiakasprojektissa ohjelmistosuunnittelutehtävissä, kehittäen edelleen samaa ohjelmistokokonaisuutta.

LÄHTEET

Agile Alliance 2013. Agile 101. Viitattu 2.9.2019, https://www.agilealliance.org/agile101/

Allman, E. 2012. Managing Technical Debt. Communications of the ACM 55 (5), 50-55.

Avison, D., Fitzgerald, G. 2002. Engineering themes. Teoksessa Information systems development. Maidenhead: McGraw-Hill Education, 151-152.

Beck, K. ym. 2001. Manifesto for agile software development. Viitattu 9.8.2019, agilemanifesto.org.

Maples, C. 2009. Enterprise Agile Transformation: The Two-Year Wall. *2009 Agile Conference*. Chicago: IEEE, 90-95.

Overview of the Scaled Agile Framework for Lean Enterprises. 2019. Viitattu 9.8.2019, Scaled Agile: https://www.scaledagile.com/resources/safe-whitepaper/.

Poppendieck, M., Poppendieck, T. 2003. Lean software development: An agile toolkit. Addison Wesley Professional.

Rodríguez, P. 2013. Combining Lean thinking and Agile Software Development. How do software-intensive companies use them in practice? Oulu: University of Oulu.

Schwaber, K. & Sutherland, J. 2016. The Scrum Guide. The Definitive Guideto Scrum: The Rules of the Game. Scrum.org. Viitattu 2.9.2019. https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf

Thomas, D. 2006. Agile Evolution Towards the Continuous Improvement of Legacy Software. Journal of Object Technology, 5 (7) 19-26.

Womack, J. P., Jones, D. T. 1996. Lean Thinking: Banish waste and create wealth in your organisation. New York: Rawson Associates.